QUEM PARTILHA O QUE SABE, MUDA A HISTÓRIA DE QUEM APRENDE.

ANO 2025

```
if (n < 2)
{
   for (int i = 2; i <= n /2; i++)
   {  return true;
   }
for (int i =1)
   for (int i = =2; <= n/2; i++)
      return false;
         }
   }
```

lógica de programação

TUDO É POSSÍVEL SE VOCÊ QUISER
NO LIMIT ME!

AUTOR KENNY VEMBA

Lógica de Programação

LA MÁFIA

Autor Kenny Vemba

Kenny Vemba é um pensador curioso, apaixonado por aprender, criar e compartilhar conhecimento. Com uma mente inquieta e um olhar atento ao mundo ao seu redor, busca constantemente evoluir, transformar ideias em ação e inspirar outras pessoas a fazerem o mesmo.

Este livro é fruto de experiências, reflexões e aprendizados ao longo de sua jornada pessoal e profissional. Kenny acredita que o conhecimento só tem valor quando compartilhado, e por isso, escreve com sinceridade, clareza e propósito.

Mais do que um autor, ele se vê como um eterno aprendiz – Alguém que escreve não para mostrar o que sabe, mas para descobrir, junto com o leito, novos caminhos e possibilidades.

Agradecimentos

Venho por está agradecer primeiramente à DEUS pela permissão da minha obra que ajudará a muitos, agradeço ainda ao Professor Fernando De Sousa pela sua criatividade nas aulas, eu já vi várias maneiras de dar aulas mas igual a sua é única, ensinando diversas coisas na minha vida da programação, eu lembro-me quando eu comecei, eu era alguém que não era o melhor ou me encontrava nesse meio, mas o Professor Fernando sempre acreditou, e tornei-me no que sou hoje e tinha uma frase de que diz :**"Primeiro cumpri, depois reclama."**, após isso agradeço a **La Mafia**, que têm sido uma grande família como é dito: La Mafia, Una Família, particularmente agradeço ao **Kaster Code** pela disponibilidade do seu computador que foi bastante útil e sempre será, não só, ajudou-me e ajuda-me bastante nas minhas práticas de programação e pentest, porque não posou um computador ou um telemóvel digital, **TroubleShoot**, esse é o cara que partilho conteúdos de programação e pentest tem sido um verdadeiro irmão, me ajuda e me apoia, **KingZaca** esse é um bro que sempre tem dado energia positiva, trazendo ideias de negócios e outros, **Itamar Dos Santos**, O homem do momento que sempre tem ideias malucas, as vezes tem ideias que você não imaginas, esse é o cara do momento, **Otiniel Manuel**, faz a correção quanto menos você imagina, onde você acha que está certo , esse cara aparece para te corrigir, **Carlos Daniel** o manager, criativo em momento, bom manager e improvisor, esse cara fala o que você menos esperas ouvir, em suma agradeço a **La Mafia**, têm sido uns verdadeiro heróis.

Índice

Prefácio....................................1

O que é Lógica de Programação....................................2

Exemplo de lógica fora da computação....................................3

Relação entre lógica e resolução de problemas....................................4

Definição de algoritmo....................................5

Formas de Representação de um Algoritmo....................................7

O que são variáveis?....................................10

Tipos de Variáveis Mais Comuns na Lógica de Programação....................................11

Por que usar tipos diferentes?....................................12

- ✓ Entrada e Saída de Dados na Lógica de Programação....................................12
- ✓ Condições Simples: IF e ELSE....................................12
- ✓ Condições Compostas: AND, OR, NOT....................................12

Estruturas de Repetição: WHILE, FOR, DO WHILE....................................18

Operadores Aritméticos....................................21

Operadores Relacionais....................................21

Operadores Lógicos....................................21

Funções na Programação: Por Que Usar e Como Funciona....................................22

Lógica em Diferentes Linguagens Como aplicar o que aprendeu em:....................................26

Dicas Para o Futuro Programador Plataformas para praticar (ex: Visualg, HackerRank, Code.org)....................................28

Glossário de Lógica de Programação....................................31

Forma de Aprendizado ou conhecendo a maneira como você aprende...**32**

Como o Cérebro Aprende?..**33**

Conclusão..**34**

Prefácio

Eu decide escrever esse livro por causa das debilidades que eu pode notar com os alunos que terminaram o médio e os alunos da 13ª classe no curso de Informática de Gestão, mas não só, estamos no mundo onde as TIs encontram-se avançadas, AI (Artificial Intelligent), onde por nada basta digitar e automaticamente trás a sua pesquisa, com isso porquê preocupar-se em pensar se basta eu digitar ou falar e tenho a minha resposta resumida e simplificada?, muitos vão questionar dessa forma porquê não compreendem o poder da Lógica de Programação ou ainda como eu a chamo "**Lógica Computacional**".

Aristóteles (384–322 a.C.): considerado o "pai da lógica", criou o **silogismo**, uma forma básica de raciocínio lógico, mas puramente teórico, ainda a muitos mais como o matemático **Gottfried Wilhelm Leibniz** imaginava uma "língua universal" baseada na lógica para resolver disputas racionais — uma ideia precursora da programação, **Boole (1854)**: criou a **Álgebra Booleana**, base matemática fundamental para a lógica computacional e os circuitos digitais modernos.

Lógica Computacional, eu defino como um algoritmo, pois têm o mesmo princípio e o mesmo fim, caminhando para resolução de um problema ou mais, sendo finito e não-ambígua.

OBS: Ela é infinita no sentido criativo, pois não há limite para o que você pode construir.

A sua importância: lógica de programação ajuda você a aprender qualquer linguagem com mais facilidade, pois os princípios (como estruturas de decisão, repetições, variáveis e funções) são comuns em todas elas. Melhora a capacidade de pensar de forma lógica e analítica, o que é útil não só na programação, mas também em outras áreas como matemática, engenharia e até no dia a dia.

Organização:

Lógica de Programação;

Forma de Aprendizado ou conhecendo a maneira como você aprende.

Conteúdo Principal

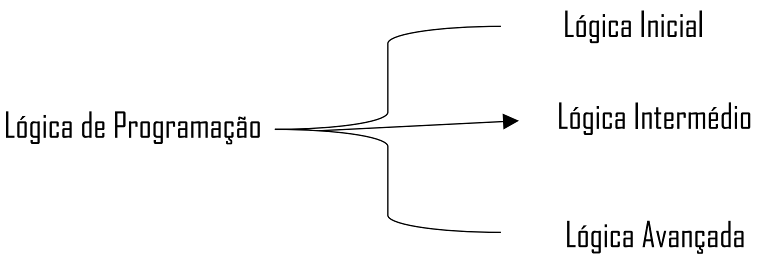
Capít.01 O que é Lógica de Programação

Lógica de Programação é a **base do raciocínio usado para criar algoritmos e resolver problemas com o uso de programação**. Ela envolve a organização de ideias e instruções de forma estruturada e lógica, para que um computador possa entender e executar tarefas passo a passo.

Em outras palavras:

É o **conjunto de regras e técnicas** que usamos para **planejar a solução de um problema** antes de escrever o código em uma linguagem de programação.

Com base nas minhas pesquisas humanas eu descrevo ou afirmo que a lógica de programação, ele tem 3 níveis de raciocínio, que são:



Lógica Inicial de Computacional é a base de raciocínio de qualquer pessoa, ela já nasce com isso, onde é intendido somente o básico.

Lógica Intermédio da Computacional é o nível onde o raciocínio da pessoa começa a ganha aceleração, onde passa de 40km/h para 80km/h, onde começa a intender as coisas passadas de maneiras simples, fáceis, mais rápido de interpretar, alterar, reparar, implementar facilmente.

Lógica Avançada da Computacional, onde o cérebro acelera de 100km/h à 180km/h, suas capacidades de processar ou usar o cérebro vai de 40% ou mais, interpreta de maneira super rápido, compreende de maneira espetacular, raciocínio 100 vezes mais rápido a cada dia que prática, seu poder de resolução é incrível.

Percentagem

Lógica Inicial vai de 0% até 20%;

Lógica Intermédio vai de 21% até 50%;

Lógica Avançada vai de 51 até 100%.

Principais elementos da lógica de programação:

- **Sequência** – execução das instruções em ordem;
- **Decisão (condicional)** – uso de estruturas como if, else, que permitem tomar decisões com base em condições;
- **Repetição (laços)** – repetição de blocos de código com estruturas como while, for;
- **Variáveis** – locais na memória para guardar valores que podem mudar;
- **Operadores lógicos e aritméticos** – usados para comparar valores e fazer cálculos.

Exemplo simples:

Problema: Verificar se uma pessoa pode votar.

Lógica de programação:

1. Receber a idade da pessoa.
2. Se a idade for maior ou igual a 16, exibir "Pode votar".
3. Senão, exibir "Não pode votar".

Importância:

- Ajuda a **organizar o pensamento lógico**;
- Facilita a **criação de programas eficientes e corretos**;
- É o **primeiro passo** para quem quer aprender a programar.

Exemplo de lógica fora da computação

A **lógica de programação** pode ser facilmente comparada a **tarefas do dia a dia**, como **fazer café** ou **seguir uma receita**. Essas atividades seguem **passos organizados**, **decisões** e até **repetições**, exatamente como um algoritmo.

Exemplo: Fazer café (algoritmo no mundo real)

Problema: Fazer uma xícara de café.

Lógica aplicada (passo a passo):

1. Coloque água na chaleira.
2. Leve ao fogo e **espere até ferver**. *(repetição: esperar até uma condição ser verdadeira)*
3. Coloque o pó de café no coador.

4. Quando a água ferver, despeje sobre o pó.
5. Espere o café coar completamente.
6. Se quiser, **adicione açúcar**. *(condição: deseja adoçar?)*
7. Sirva na xícara.
8. Beba com cuidado.

Outro exemplo: Seguir uma receita de macarrão

1. Ferva água numa panela.
2. Quando ferver, adicione o macarrão.
3. Cozinhe por 10 minutos ou até ficar al dente. *(repetição com tempo ou teste de condição)*
4. Escorra a água.
5. Se quiser molho, prepare o molho.
6. Misture e sirva.

Comparação com lógica de programação:

Vida real	**Programação**
Lista de passos (receita)	Sequência de comandos
"Se quiser açúcar..."	Estrutura condicional if
"Espere ferver..."	Laço de repetição while
Ingredientes variáveis	Uso de variáveis

Esses exemplos mostram que usamos lógica o tempo todo — programar é só aplicar essa lógica de forma estruturada para que o computador entenda.

Relação entre lógica e resolução de problemas

A **relação entre lógica e resolução de problemas** é fundamental, especialmente em áreas como programação, matemática, engenharia — mas também em situações cotidianas.

Como a lógica se relaciona com a resolução de problemas:

1. **Análise do problema**

- A lógica ajuda a **entender a situação**, identificar **o que está sendo pedido** e quais são os dados disponíveis.
- Exemplo: "Preciso saber se um número é par ou ímpar."

2. **Planejamento da solução**

    - Com lógica, organizamos os **passos necessários** para chegar a uma solução.
    - Exemplo: "Divida o número por 2, veja se o resto é 0."

3. **Tomada de decisão**

    - A lógica permite **escolher caminhos diferentes** com base em condições.
    - Exemplo: "Se chover, leve guarda-chuva; senão, vá sem ele."

4. **Execução e correção**

    - A lógica também permite **testar soluções**, identificar erros e corrigir o raciocínio.

Exemplo prático:

Problema: "Como saber se alguém pode comprar um ingresso com desconto?"

Raciocínio lógico:

- Dados: idade da pessoa.
- Regra: pessoas com menos de 18 ou mais de 60 têm direito a desconto.
- Solução:
    - Se a idade < 18 **ou** idade > 60 → aplicar desconto.
    - Senão → preço normal.

Conclusão:

A lógica:

- **Estrutura o pensamento;**
- **Evita soluções aleatórias ou confusas;**
- **Permite generalizar soluções** para problemas semelhantes;
- É a **base para algoritmos eficientes.**

Definição de algoritmo

Um **algoritmo** é um **conjunto de passos finitos, organizados e lógicos** usados para **resolver um problema ou realizar uma tarefa específica**.

Características principais de um algoritmo:

1. **Finito** – deve terminar após um número limitado de etapas.
2. **Bem definido** – cada passo deve ser claro, sem ambiguidade.
3. **Ordenado** – os passos devem estar na sequência correta.
4. **Efetivo** – cada instrução deve ser executável na prática.

Exemplo simples (algoritmo para somar dois números):

1. Leia o primeiro número.
2. Leia o segundo número.
3. Some os dois números.
4. Mostre o resultado.

Etimologia:

A palavra **algoritmo** vem do nome do matemático persa **Al-Khwarizmi**, que viveu no século IX e escreveu sobre métodos de cálculo.

Aplicações:

- Resolver contas matemáticas
- Programar computadores
- Criar rotinas de máquinas
- Automatizar decisões

Como escrever um algoritmo simples

Escrever um algoritmo envolve **definir uma sequência de passos lógicos e organizados** para resolver um problema. Você não precisa, necessariamente, usar uma linguagem de programação para isso — pode usar **linguagem natural**, **pseudocódigo** ou **fluxogramas**.

Passos básicos para escrever um algoritmo simples:

1. **Entenda o problema**
    - O que você precisa resolver?
2. **Identifique os dados de entrada**
    - Quais informações serão fornecidas?

3. **Defina o que precisa ser feito (processamento)**
    - Quais operações ou decisões precisam ser realizadas?
4. **Determine a saída esperada**
    - O que deve ser mostrado no final?

Exemplo de algoritmo simples: "Descobrir se uma pessoa é maior de idade"

Em linguagem natural:

1. Ler a idade da pessoa;
2. Se a idade for maior ou igual a 18, mostrar "Maior de idade";
3. Senão, mostrar "Menor de idade".

Em pseudocódigo:

```
Início
 Leia idade
 Se idade >= 18 então
   Escreva "Maior de idade"
 Senão
   Escreva "Menor de idade"
Fim
```

Estrutura usada:

- **Leitura (entrada)**: Leia idade
- **Decisão (condicional)**: Se... então... senão
- **Saída**: Escreva

Formas de Representação de um Algoritmo

Um mesmo algoritmo pode ser representado de **diferentes maneiras**, dependendo do objetivo e do público. As três formas mais comuns são:

Texto (Linguagem Natural)

- Usa frases simples, como uma receita ou instruções do dia a dia.
- Boa para iniciantes, mas pode ser ambígua se não for bem estruturada.

Exemplo:

Problema: Verificar se uma pessoa pode votar.

1. Leia a idade da pessoa.
2. Se a idade for maior ou igual a 16, informe que pode votar.
3. Caso contrário, informe que não pode votar.

Pseudocódigo

- Usa uma linguagem estruturada semelhante à programação, mas mais simples.
- Facilita a transição para linguagens de programação reais.
- Não segue uma sintaxe fixa, mas sim lógica clara.

Exemplo:

```
Início
  Leia idade
  Se idade >= 16 então
     Escreva "Pode votar"
  Senão
     Escreva "Não pode votar"
Fim
```

Fluxograma

- Representação **visual** com símbolos gráficos.
- Mostra o **fluxo de execução** do algoritmo.
- Útil para entender a lógica rapidamente.

Principais símbolos:

Símbolo	Função
▣ Retângulo	Processo (ação)
⬭ Losango	Decisão (condição)
■ Paralelepípedo	Entrada/Saída
● Círculo	Início/Fim

Exemplo (resumo visual):

[Início]

↓

[Leia idade]

↓

[idade ≥ 16?] → Sim → [Escreva "Pode votar"]

↓

Não

↓

[Escreva "Não pode votar"]

↓

[Fim]

Conclusão:

Forma	Melhor para...
Texto	Explicações simples e diretas
Pseudocódigo	Planejar algoritmos antes do código
Fluxograma	Visualizar a lógica do processo

O que são variáveis?

Em programação (e na lógica de programação), **variáveis são espaços na memória usados para armazenar valores que podem mudar durante a execução do programa**.

Explicando de forma simples:

Uma **variável é como uma "caixinha com um nome"**, onde você guarda uma informação que pode ser consultada, alterada ou usada em cálculos.

Exemplo prático:

idade ← 18

Aqui:

idade é o **nome da variável**.

18 é o **valor armazenado**.

Você pode:

- **Atribuir** um valor a ela:
  nome ← "Lucas"
- **Usar seu valor em um cálculo**:
  dobro ← idade * 2
- **Atualizar o valor**:
  idade ← idade + 1

Por que variáveis são importantes?

- Guardam **informações temporárias** durante a execução do programa.
- Permitem **personalizar a lógica** de acordo com diferentes dados.
- Tornam o código **mais flexível e reutilizável**.

Tipos comuns de variáveis (dependem da linguagem de programação):

Tipo	**Exemplo de valor**	**Representa**
Inteiro	20	Números inteiros
Real/Float	3.14	Números com casas decimais
Texto	"Maria"	Palavras ou frases (strings)

Tipo	Exemplo de valor	Representa
Booleano	Verdadeiro / Falso	Respostas lógicas (sim/não)

Tipos de Variáveis Mais Comuns na Lógica de Programação

Na lógica e nas linguagens de programação, os **tipos de variáveis** determinam **que tipo de dado** pode ser armazenado. Usar o tipo correto é essencial para garantir que o algoritmo funcione corretamente.

1.Inteiro (int)

- Armazena números **inteiros**, ou seja, **sem casas decimais**.
- Ex: -5, 0, 42, 2025

idade ← 18

2.Real (float ou double)

- Armazena números **decimais** (com vírgula/ponto flutuante).
- Ex: 3.14, 0.5, -7.89

peso ← 65.5

3.Caractere (char) ou Texto (string)

- **Caractere (char):** armazena **um único símbolo** (letra, número ou sinal). Ex: 'A', '1', '@'
- **Texto (string):** armazena uma **sequência de caracteres** (palavras ou frases). Ex: "Maria", "Olá, mundo!"

letra ← 'B'

nome ← "Lucas"

4.Booleano (bool)

- Armazena apenas dois valores: **Verdadeiro** ou **Falso**.
- Usado em **decisões lógicas**.

maiorDeIdade ← verdadeiro

Por que usar tipos diferentes?

Cada tipo tem um **comportamento próprio** e é usado em **situações específicas**:

- Cálculos → inteiros e reais.
- Textos e nomes → strings.
- Decisões (condições) → booleanos.

Entrada e Saída de Dados na Lógica de Programação

Na lógica de programação, **entrada e saída de dados** são essenciais para a comunicação entre o **usuário** e o **algoritmo**.

1. Entrada de Dados (INPUT)

É quando o **algoritmo recebe dados do usuário** ou de outra fonte para processar.

Exemplos:

- Ler a idade de uma pessoa
- Receber um nome
- Digitar um número para somar

Em pseudocódigo:

```
Leia idade
Leia nome
```

2. Saída de Dados (OUTPUT)

É quando o **algoritmo exibe uma resposta** ou resultado para o usuário.

Exemplos:

- Mostrar o nome da pessoa
- Exibir o resultado de um cálculo
- Informar se a pessoa pode votar

Em pseudocódigo:

```
Escreva "Seu nome é", nome
Escreva "Sua idade é", idade
```

Exemplo completo:

```
Início
  Leia nome
  Leia idade
  Escreva "Olá, ", nome
  Escreva "Você tem ", idade, " anos"
Fim
```

Resumo:

Ação	Comando comum (pseudocódigo)	O que faz
Entrada	Leia	Recebe dados do usuário
Saída	Escreva	Mostra informações na tela

Condições Simples: IF e ELSE

Na lógica de programação, as **condições** são usadas para **tomar decisões**, ou seja, para executar **ações diferentes dependendo de uma situação**. Os comandos mais básicos são:

IF ("se")

Executa um bloco de código **se uma condição for verdadeira**.

Sintaxe básica (pseudocódigo):

```
Se condição então
  [ações]
```

IF...ELSE ("se...senão")

Permite **duas possibilidades**: o que fazer **se a condição for verdadeira** e **o que fazer se for falsa**.

Sintaxe básica:

Se condição então

  [ações se verdadeiro]

Senão

  [ações se falso]

Exemplo prático:

Problema: Verificar se uma pessoa é maior de idade.

```
Início
  Leia idade
  Se idade >= 18 então
    Escreva "Maior de idade"
  Senão
    Escreva "Menor de idade"
Fim
```

Como funciona:

- O algoritmo **testa a condição** (idade >= 18).
- Se for **verdadeiro**, executa o bloco do IF.
- Se for **falso**, executa o bloco do ELSE.

Operadores lógicos comuns em condições:

Operador	Significado	Exemplo
==	Igual a	idade == 18
!=	Diferente de	idade != 18
>	Maior que	idade > 18
<	Menor que	idade < 18
>=	Maior ou igual a	idade >= 18
<=	Menor ou igual a	idade <= 18

Condições Compostas: AND, OR, NOT

As **condições compostas** permitem **combinar duas ou mais condições lógicas** em uma única verificação, usando os operadores **lógicos**:

1. AND (E)

- Todas as condições devem ser **verdadeiras** para o resultado final ser verdadeiro.
- **Símbolo:** AND

Exemplo:

Se idade >= 18 AND idade <= 60 então

  Escreva "Voto obrigatório"

Só executa se a **idade for entre 18 e 60**.

2. OR (OU)

- Basta **uma das condições** ser verdadeira para o resultado final ser verdadeiro.
- **Símbolo:** OR

Exemplo:

Se idade < 16 OR idade > 70 então

  Escreva "Voto facultativo"

Executa se a pessoa **for menor de 16 ou maior de 70**.

3. NOT (NÃO)

- **Inverte** o valor lógico de uma condição.
- **Símbolo:** NOT

Exemplo:

Se NOT chuva então

  Escreva "Pode sair sem guarda-chuva"

Executa **se não estiver chovendo**.

Exemplo completo com várias condições:

```
Início
  Leia idade
  Se idade >= 18 AND idade <= 60 então
    Escreva "Voto obrigatório"
  Senão Se idade < 16 OR idade > 70 então
    Escreva "Voto facultativo"
  Senão
    Escreva "Voto opcional"
Fim
```

Tabela resumo:

Operador	**Nome**	**Retorna Verdadeiro se...**
AND	E	**Todas** as condições forem verdadeiras
OR	OU	**Pelo menos uma** condição for verdadeira
NOT	NÃO	A condição for **falsa** (e será invertida)

Exemplos aplicados

Aqui vão **exemplos práticos aplicando condições simples e compostas**, com entradas, saídas e uso de variáveis — tudo dentro da lógica de programação.

Exemplo 1 – Condição Simples (IF / ELSE)

Problema: Verificar se um número é par ou ímpar.

```
Início
  Leia numero
  Se numero % 2 == 0 então
    Escreva "Número par"
  Senão
    Escreva "Número ímpar"
```

Fim

Exemplo 2 – Condição Composta (AND)

Problema: Verificar se a pessoa tem idade para votar obrigatoriamente (entre 18 e 60 anos).

```
Início
  Leia idade
  Se idade >= 18 AND idade <= 60 então
    Escreva "Voto obrigatório"
  Senão
    Escreva "Voto não obrigatório"
Fim
```

Exemplo 3 – Condição Composta (OR)

Problema: Verificar se uma pessoa tem direito a meia-entrada (estudante ou maior de 60 anos).

```
Início
  Leia estudante
  Leia idade
  Se estudante == verdadeiro OR idade >= 60 então
    Escreva "Tem direito à meia-entrada"
  Senão
    Escreva "Não tem direito à meia-entrada"
Fim
```

Exemplo 4 – Condição com NOT

Problema: Mostrar mensagem só se o usuário **não** estiver logado.

```
Início
  Leia logado
  Se NOT logado então
```

Escreva "Por favor, faça login"

Fim

Esses exemplos usam:

- **Variáveis** (idade, numero, logado)
- **Entradas** (Leia)
- **Saídas** (Escreva)
- **Condicionais** (Se, Senão)
- **Operadores lógicos** (AND, OR, NOT)

Estruturas de Repetição: WHILE, FOR, DO WHILE

As **estruturas de repetição** (ou laços) são usadas para **executar blocos de código várias vezes**, de forma automática, até que uma condição seja satisfeita.

ENQUANTO (WHILE)

Executa o bloco de código **enquanto a condição for verdadeira**.

Sintaxe (pseudocódigo):

```
Enquanto condição faça
  [ações]
FimEnquanto
```

Exemplo:

```
contador ← 1
Enquanto contador <= 5 faça
  Escreva contador
  contador ← contador + 1
FimEnquanto
```

PARA (FOR)

Usado quando **sabemos exatamente quantas vezes** o laço deve repetir.

Sintaxe:

Para i de 1 até 5 faça

  [ações]

FimPara

Exemplo:

Para i de 1 até 5 faça

  Escreva i

FimPara

REPITA ATÉ (DO WHILE)

Executa o bloco **pelo menos uma vez**, e **repete até que a condição se torne verdadeira**.

Sintaxe:

Repita

  [ações]

Até condição

Exemplo:

contador ← 1

Repita

  Escreva contador

  contador ← contador + 1

Até contador > 5

Comparativo rápido:

Estrutura	Quando usar
ENQUANTO	Quando **não sabemos** quantas repetições.
PARA	Quando sabemos **quantas vezes repetir.**
REPITA ATÉ	Quando queremos que execute **ao menos uma vez.**

Exemplos práticos e comuns

Exemplos práticos e comuns usando as três principais **estruturas de repetição**: ENQUANTO (WHILE), PARA (FOR), e REPITA ATÉ (DO WHILE). São situações do dia a dia que ajudam a entender o uso de cada uma.

1. WHILE – Ler notas até uma nota inválida

```
Leia nota
Enquanto nota >= 0 E nota <= 10 faça
 Escreva "Nota válida:", nota
 Leia nota
FimEnquanto
Escreva "Nota inválida. Encerrando."
```

2. FOR – Mostrar tabuada do 7

```
Para i de 1 até 10 faça
 Escreva "7 x", i, "=", 7 * i
FimPara
```

3. DO WHILE – Repetir até digitar “sair”

```
Repita
 Leia comando
 Escreva "Você digitou:", comando
Até comando == "sair"
Escreva "Programa finalizado"
```

Outros exemplos comuns:

Situação	Estrutura ideal
Verificar senhas até acertar	DO WHILE
Contar de 1 a 100	FOR
Esperar uma condição mudar	WHILE
Ler lista de nomes	WHILE ou FOR
Mostrar menu até sair	DO WHILE

Operadores Aritméticos

Usados para fazer **cálculos matemáticos**.

Operador	Significado	Exemplo	Resultado
+	Soma	5 + 3	8
-	Subtração	10 - 4	6
*	Multiplicação	2 * 6	12
/	Divisão	10 / 2	5
%	Resto da divisão	7 % 2	1

Operadores Relacionais

Usados para **comparar valores**. Retornam **Verdadeiro** ou **Falso**.

Operador	Significado	Exemplo	Resultado
==	Igual a	5 == 5	Verdadeiro
!=	Diferente de	5 != 3	Verdadeiro
>	Maior que	7 > 2	Verdadeiro
<	Menor que	4 < 1	Falso
>=	Maior ou igual a	5 >= 5	Verdadeiro
<=	Menor ou igual a	3 <= 2	Falso

Operadores Lógicos

Usados para **combinar condições**. Trabalham com valores **booleanos** (Verdadeiro/Falso).

Operador	Nome	Exemplo	Resultado
AND	E	idade >= 18 AND idade <= 60	Verdadeiro se **ambas** forem verdadeiras
OR	OU	nota < 0 OR nota > 10	Verdadeiro se **uma** for verdadeira
NOT	NÃO	NOT chovendo	Inverte o valor (ex: de verdadeiro para falso)

Exemplo Prático com os 3 tipos:

```
Leia idade
Se idade >= 18 AND idade <= 60 então
  Escreva "Voto obrigatório"
Senão
  Escreva "Voto não obrigatório"
```

- >= e <= → **relacionais**
- AND → **lógico**
- Nenhum operador aritmético aqui, mas poderia ser usado, por exemplo: idade + 1

Funções na Programação: Por Que Usar e Como Funciona

Por que usar funções?

Funções são **blocos de código que executam tarefas específicas**. Usá-las traz várias vantagens:

Vantagens:

- **Organização:** o código fica mais limpo e dividido em partes lógicas.
- **Reutilização:** você pode chamar a mesma função várias vezes, sem repetir código.
- **Facilidade de manutenção:** se algo mudar, você só precisa ajustar a função.
- **Leitura clara:** ajuda a entender o que cada parte do código faz.

Reutilização e Organização do Código

Sem função:

```
Escreva "Olá, Maria"
Escreva "Olá, João"
Escreva "Olá, Ana"
Com função:
Função saudacao(nome)
  Escreva "Olá, ", nome
FimFunção

saudacao("Maria")
saudacao("João")
saudacao("Ana")
```

Estrutura básica de uma função (em pseudocódigo)

```
Função nomeDaFuncao(parâmetros)

  [ações]

  Retorne valor (se necessário)

FimFunção
```

Exemplo 1: Função sem retorno

```
Função mostrarMensagem()

  Escreva "Bem-vindo!"

FimFunção
```

Exemplo 2: Função com parâmetro e retorno

```
Função quadrado(numero)

  Retorne numero * numero

FimFunção


resultado ← quadrado(5)

Escreva "Resultado: ", resultado
```

Quando usar funções?

Situação	**Use função?**
Repetir um cálculo	✅ Sim
Organizar partes do algoritmo	✅ Sim
Executar ações com base em dados	✅ Sim

Praticando com Problemas Reais

praticar a lógica de programação com problemas reais — com **exercícios comentados** e **soluções passo a passo** usando variáveis, condições, laços e funções.

Exercício 1: Calcular a média de duas notas

Enunciado:

Ler duas notas e exibir a média. Se a média for maior ou igual a 6, mostrar "Aprovado", senão "Reprovado".

Solução passo a passo (pseudocódigo):

```
Início
  Leia nota1
  Leia nota2
  media ← (nota1 + nota2) / 2

  Escreva "Média:", media

  Se media >= 6 então
    Escreva "Aprovado"
  Senão
    Escreva "Reprovado"
Fim
```

Exercício 2: Verificar o maior entre dois números

Enunciado:

Ler dois números e mostrar o maior deles. Se forem iguais, mostrar "Iguais".

Solução:

```
Início
  Leia num1
  Leia num2

  Se num1 > num2 então
    Escreva "Maior:", num1
  Senão Se num2 > num1 então
    Escreva "Maior:", num2
  Senão
    Escreva "Números iguais"
Fim
```

Exercício 3: Imprimir os números de 1 a 10 com laço FOR

```
Início
  Para i de 1 até 10 faça
    Escreva i
  FimPara
Fim
```

Desafio 1: Tabuada de um número

Enunciado:

Ler um número e mostrar a tabuada de 1 a 10.

Solução:

```
Início
  Leia numero
  Para i de 1 até 10 faça
    Escreva numero, " x ", i, " = ", numero * i
  FimPara
Fim
```

Desafio 2: Repetir até digitar “sair”

Enunciado:

Criar um programa que continue lendo comandos até o usuário digitar “sair”.

Solução:

```
Início
  Repita
    Leia comando
    Escreva "Comando digitado:", comando
  Até comando == "sair"
Fim
```

Desafio extra com função: Calcular fatorial

```
Função fatorial(n)
  resultado ← 1
  Para i de 1 até n faça
    resultado ← resultado * i
  FimPara
  Retorne resultado
FimFunção
```

```
Início
  Leia numero
  Escreva "Fatorial:", fatorial(numero)
Fim
```

Lógica em Diferentes Linguagens Como aplicar o que aprendeu em:

- Portugol
- Python
- JavaScript
- C/C++

A lógica de programação é a **base comum** entre todas as linguagens. O que muda é a **sintaxe** — ou seja, a forma de escrever os comandos.

Vamos comparar como aplicar a **mesma lógica** (exemplo: calcular a média de duas notas e mostrar "Aprovado" ou "Reprovado") em:

- 🟦 **Portugol**
- 🐍 **Python**
- 🌐 **JavaScript**
- 💻 **C/C++**

Problema exemplo:

Ler duas notas, calcular a média e informar se o aluno está aprovado (média ≥ 6) ou reprovado.

Portugol (pseudocódigo visual)

```
Algoritmo "media"
Var
  nota1, nota2, media: real
Inicio
  Escreva("Digite a primeira nota: ")
  Leia(nota1)
  Escreva("Digite a segunda nota: ")
  Leia(nota2)

  media ← (nota1 + nota2) / 2

  Se media >= 6 então
```

```
    Escreva("Aprovado")
  Senão
    Escreva("Reprovado")
FimAlgoritmo
```

Python

```python
nota1 = float(input("Digite a primeira nota: "))
nota2 = float(input("Digite a segunda nota: "))

media = (nota1 + nota2) / 2

if media >= 6:
    print("Aprovado")
else:
    print("Reprovado")
```

JavaScript (em navegador ou Node.js)

```javascript
let nota1 = parseFloat(prompt("Digite a primeira nota:"));
let nota2 = parseFloat(prompt("Digite a segunda nota:"));

let media = (nota1 + nota2) / 2;

if (media >= 6) {
  alert("Aprovado");
} else {
  alert("Reprovado");
}
```

C / C++

```c
#include <stdio.h>

int main() {
    float nota1, nota2, media;

    printf("Digite a primeira nota: ");
    scanf("%f", &nota1);

    printf("Digite a segunda nota: ");
```

```c
    scanf("%f", &nota2);

    media = (nota1 + nota2) / 2;

    if (media >= 6) {
        printf("Aprovado\n");
    } else {
        printf("Reprovado\n");
    }

    return 0;
}
```

Comparativo Rápido

Lógica	**Igual em todas**
Leitura de dados	Sim
Operações aritméticas	Sim
Condicionais (IF / ELSE)	Sim
Estrutura de decisão	Sim
Diferença	Só a **sintaxe** (forma de escrever)

Dicas Para o Futuro Programador Plataformas para praticar (ex: Visualg, HackerRank, Code.org)

Como continuar aprendendo

Reforço da importância da lógica na vida profissional

Aqui estão **dicas práticas e valiosas para quem está começando na programação**, com sugestões de plataformas, formas de continuar aprendendo e um reforço sobre a importância da lógica.

Dicas Para o Futuro Programador

1. A lógica é a base de tudo

Antes de se preocupar com linguagens como Python, Java ou C++, aprenda **a pensar como um programador**:

- Dividir um problema em partes menores
- Criar passos lógicos para resolver algo
- Antecipar erros e prever cenários

☞ **Quem domina lógica aprende qualquer linguagem com mais facilidade.**

2. Plataformas para praticar programação e lógica

Iniciantes (visual, intuitivo):

- **Visualg** – Ideal para treinar lógica com pseudocódigo (Portugol)
- **Code.org** – Gamificado e ótimo para começar (até crianças usam!)
- **Scratch** – Programação com blocos (visual)

Programadores iniciantes/intermediários:

- **HackerRank** – Desafios de lógica, algoritmos e estruturas de dados
- **Beecrowd** (antigo URI) – Muito usado em faculdades
- **Exercism.io** – Suporte com mentores, código em várias linguagens
- **Replit** – Para codar direto do navegador, sem instalar nada

3. Como continuar aprendendo

- Estude **resolvendo problemas** reais (não apenas assistindo aulas)
- Pratique todo dia, mesmo que seja **15 minutos**
- Leia códigos de outras pessoas (aprendemos muito assim!)
- Participe de **comunidades** (fóruns, grupos de estudo, GitHub)
- Aprenda uma linguagem popular (Python é ótima para começar)

Dica: **Não se preocupe em decorar, se preocupe em entender o raciocínio por trás.**

4.A importância da lógica na vida profissional

A lógica está em **toda profissão de tecnologia**: desenvolvedor, analista, cientista de dados, engenheiro de software etc.

Profissionais com boa lógica resolvem problemas com **clareza, eficiência e criatividade**

Mesmo fora da programação, lógica ajuda a:

- Tomar decisões melhores
- Automatizar tarefas

- Pensar de forma estruturada
- Quem tem boa lógica se destaca no mercado de trabalho e aprende mais rápido novas tecnologias.

Resumo final para iniciantes:

✔ Comece pela lógica

✔ Use plataformas simples como Visualg, Code.org

✔ Evolua para Python, JavaScript ou C

✔ Pratique todos os dias com pequenos desafios

✔ Nunca pare de aprender

Apêndice / Glossário

Termos técnicos explicados de forma simples

Aqui está um **apêndice/glossário simples e direto** com os **termos técnicos mais comuns na lógica de programação**, explicados de forma acessível:

Glossário de Lógica de Programação

Termo	**Significado Simples**
Algoritmo	Conjunto de passos para resolver um problema. Ex: receita de bolo.
Variável	Um "espaço" na memória para guardar informações (como uma caixinha com nome).
Constante	Parecida com a variável, mas **não pode mudar** de valor depois de definida.
Entrada	Dados que o programa recebe do usuário (ex: input).
Saída	Informações que o programa mostra para o usuário (ex: print).
Operador	Símbolo que realiza uma operação. Ex: +, >, AND.
Condicional (IF)	Verifica se uma condição é verdadeira para tomar uma decisão.
Laço de repetição	Comando que **repete** um bloco de código várias vezes. Ex: FOR, WHILE.
Função	Bloco de código que executa uma tarefa específica. Pode ser reutilizado.
Booleano	Tipo de dado que só aceita dois valores: **verdadeiro** ou **falso**.
Pseudocódigo	Uma forma intermediária entre linguagem natural e linguagem de programação.
Sintaxe	As "regras de escrita" da linguagem. Cada linguagem tem a sua.
Depuração (Debug)	Ato de encontrar e corrigir erros no código.
Compilar	Traduzir o código-fonte para linguagem de máquina.
Loop infinito	Quando o código entra em um laço que **nunca termina** (geralmente erro).
Comentário	Texto no código que **não é executado**, usado para explicar partes do código.

Dica final:

Se você entender **o que cada termo representa na prática**, sua jornada como programador será muito mais tranquila.

Forma de Aprendizado ou conhecendo a maneira como você aprende

Como o Cérebro Aprende?

Você já se perguntou por que algumas coisas você aprende mais rápido do que outras?

A resposta está na forma como você interage com o conteúdo. Segundo esta representação visual, o cérebro retém muito mais informação quando você participa ativamente do processo de aprendizagem. Veja só:

10% Lendo: Aprendizado passivo. Ótimo para introdução, mas limitado para memorização duradoura.

20% Ouvindo: Melhora um pouco, mas ainda é um processo receptivo.

30% Observando: Ver alguém fazendo algo ativa mais áreas do cérebro.

50% Vendo e Ouvindo: Combinar estímulos visuais e auditivos já ajuda bastante na retenção.

70% Discutindo com os outros: A interação e o debate provocam reflexão e fixação do conteúdo.

80% Fazendo: A prática é um dos métodos mais eficazes de aprendizado. "Aprende-se fazendo".

95% Ensinando os outros: O topo da pirâmide! Ensinar exige que você compreenda profundamente o conteúdo — e isso solidifica o conhecimento.

Conclusão: Quer aprender de verdade?. Vá além da leitura e da escuta. Pratique, converse, ensine. O conhecimento se fortalece quando é colocado em acção.

Conclusão

A lógica de programação é a base para criar algoritmos, ou seja, sequências de passos organizados para resolver problemas. Ela envolve o uso de raciocínio lógico, estruturas como condições, laços e variáveis, e o entendimento de como o computador executa comandos. Aprender lógica de programação pode parecer desafiador no início, mas se torna mais fácil quando usamos diferentes formas de aprendizado:

Lendo: livros, apostilas e tutoriais ajudam a entender conceitos teóricos e a linguagem usada na programação.

Ouvindo: podcasts, aulas e explicações verbais reforçam a compreensão, principalmente para quem aprende melhor por meio da escuta.

Praticando: é uma das formas mais eficazes — escrever código, errar, corrigir e testar é essencial para desenvolver habilidade lógica.

Observando: ver outras pessoas programando ou resolvendo problemas ajuda a entender estratégias e boas práticas.

Vendo e ouvindo: vídeos e aulas gravadas combinam som e imagem, o que facilita o aprendizado visual e auditivo.

Discutindo: trocar ideias com colegas ou professores ajuda a esclarecer dúvidas e ver problemas por outros pontos de vista.

Ensinando: explicar o que aprendeu para outra pessoa é uma das melhores formas de fixar o conhecimento e perceber o que ainda precisa melhorar.

Cada pessoa aprende de forma diferente, por isso, combinar várias dessas estratégias é o caminho ideal para dominar a lógica de programação.